# Fișa 08 C# – membri statici, supraîncărcare metode

1. În limbajul C#, în mod implicit, membrii unei clase se pot accesa doar prin intermediul unui obiect din clasa respectivă (ei se construiesc, se alocă memorie pentru ei doar în momentul în care se creează un obiect). Există și situații în care se dorește ca o metodă sau o variabilă membru să fie accesibilă fără a fi definit un obiect, de exemplu în cazul unor funcții matematice sau a definirii unor constante. Limbajul C# permite definirea unor astfel de metode sau câmpuri folosind cuvântul cheie **static**. Exemplu:

class Test

{

//membri statici

public static int xs;

public static void fs()

{

Console.Write("\nApel functia statica.");

}

//membri ne-statici

public int x;

public void f()

{

Console.Write("\nApel functia ne-statica.");

}

}

class Program

{

static void Main(string[] args)

{

//folosire membri din clasa Test - fără a defini un obiect de tip Test

Test.xs = 7;

Test.fs();

Test.x = 2; => EROARE: trebuie definit un obiect pt. folosirea membrilor ne-statici

Test.f(); => EROARE: trebuie definit un obiect pt. folosirea membrilor ne-statici

//folosire membri din clasa Test - construind un obiect de tip Test

Test t = new Test();

t.xs = 7; => EROARE: membrilor statici nu se pot accesa prin intermediul unui obiect

t.fs(); => EROARE: membrilor statici nu se pot accesa prin intermediul unui obiect

t.x = 2;

t.f();

//gata

Console.Write("\n\n\n");

}

}

1. **Supraîncărcarea metodelor** este procedeul prin care se definesc într-o clasă mai multe metode cu același nume. În acest caz metodele trebuie să difere între ele prin numărul, poziția și / sau tipul parametrilor. În momentul apelului programul va alege automat care metodă se execută în funcție de numărul, tipul și poziția parametrilor efectivi. Acest mecanism este foarte important și formează o parte a unei facilități importante a programării obiectuale cunoscute sub numele de **polimorfism** *(polimorfism = proprietatea unui obiect sau a unei ființe de a se prezenta în mai multe forme, în funcție de condițiile exterioare)*. Vom avea astfel, aparent, o funcție care se comportă diferit în funcție de tipul, numărul sau tipul parametrilor din momentul apelului. Exemplu:

class Test

{

public void afisare() { Console.Write("\nApel 1 : fără parametri."); }

public void afisare(int i) { Console.Write("\nApel 2 : un parametru int = " + i); }

public void afisare(char c) { Console.Write("\nApel 3 : un parametru char = " + c); }

public void afisare(int i, char c) {

Console.Write("\nApel 4 : 2 parametri: int = " + i + " și char = " + c); }

}

class Program

{

static void Main(string[] args)

{

Test t = new Test();

//pentru afișare diacritice în consolă

Console.OutputEncoding = System.Text.UTF8Encoding.UTF8;

Console.Write("\nTest diacritice : Ă ă Â â Î î Ș ș Ț ț ");

//mai multe apeluri ale metodei afișare, cu parametri efectivi diferiți

//aparent metoda se comportă diferit în funcție de parametrii primiți

//(de fapt se execută variante diferite ale metodei în funcție de aceștia)

t.afisare();

t.afisare(1);

t.afisare('a');

t.afisare(1, 'a');

//t.afisare('a', 1); //EROARE => nu există o formă a metodei cu doi parametri (char, int)

//în această ordine

//gata

Console.Write("\n\n\n");

}

}

1. Rezolvați minim 5 probleme din variantele de bac2009 (subiectul II, problema 5 sau subiectul III, ultima problemă) folosind metode statice și ne‑statice. *(punct în plus la fiecare 5 probleme rezolvate, cu condiția să fie diferite de cele rezolvate de* ***oricare*** *alt coleg)*